Revisiting Approximate Top-K Algorithms in IoT

Ruifan Yang, Zheng Zhou, Lewis Tseng
Boston College
{yangrk, zhoupt, lewis.tseng}@bc.edu

XXX
XXX

***Abstract*—**

Keywords –

I. Introduction

Top-k queries have wide applications in a big data setting. One of the simplest and easy-to-grasp application is *election* which precisely capture the notion of "global" and "local" counts of votes. In a large-scale election, each ballot box has a *local* count of votes for each candidate (or item); however, what we really care about is the *global* counts of votes, i.e., the aggregation of all the local votes. More precisely, a top-k query should find the answer to the following question:

$\boxed{\text{Which are the } k \text{ globally most popular candidates?}}$

This has many natural applications in large-scale big data systems. For example, find most popular files in a peer-to-peer file-sharing system (such as BitTorrent), identify potential DoS (Denial-of-Service) attack, and produce ranking in multimedia systems [8]. Due to its wide applications, efficient Top-k mechanisms have been a hot research topic, e.g., [7], [4], [1], [5], [12], [17], [3], [8].

In a large-scale network, e.g., sensor networks, Top-k queries can be viewed as a special case of aggregation [9], [11]. Typically, the query algorithms begin with the construction of a spanning tree, and then the aggregation tasks (such as counting and summing) can be completed with $O(\log n)$ bits per node by routing data over the spanning tree. Here, n is the number of nodes in the network [14]. If n is too large, then finding "exactly" k most popular items (or candidates) has a prohibitively high communication cost. With a slight abuse of terminology, we will use item and candidate interchangeably.

To reduce the cost, approximate queries (or aggregation) mechanisms have been proposed, e.g., [13], [15], [14], [16], which do *not* always output the k most popular items – instead, some randomized tricks are used to reduce the communication costs, and the approximate mechanisms provide a guarantee (lower bound) on the amount of errors, i.e., approximation bound. However, these mechanism still require $O(k)$ communication bits per node.

In 2014, Deolalikar and Eshghi proposed a lightweight mechanism, namely *Lottery Algorithm*, to find top k items in an *approximate* fashion in distributed systems [6]. One desirable feature of the Lottery Algorithm is that each node only needs to communicate <u>a constant number of bits</u>. However, their algorithm has two limitations: (i) the output items of the Lottery Algorithm may contain a very unpopular item, i.e., no guarantee on the quality of final output is provided in [6]; and (ii) the Lottery Algorithm does not work in sparse communication networks, since the Lottery Algorithm may not converge in its current specification.

In this paper, we first formally analyze the Lottery Algorithm in [6], and address the two limitations by proposing a modular algorithm. The first stage of the algorithm selects a constant number of "popular" items in a similar fashion of the Lottery Algorithm [6], and the second stage uses a modular approach to output the Top-k items from the popular items selected in the first stage. One salient feature of the modular approach is that depending on the characteristics of the data, we can plug in different existing Top-k algorithms into the second stage to produce the output to obtain better approximation bound. This is particularly beneficial if we have some prior knowledge on the characteristics of the data, e.g., distribution of local counts of votes or communication graph. Our approximate algorithm is also lightweight, works in sparse networks and has better approximation bound.

II. Preliminary

A. System Model

The system consists of n nodes, and the communication network is modeled as a directed graph where nodes are able to communicate directly and reliably with their neighbors. The graph is assumed to be strongly connected so that effectively, each pair of nodes is able to communicate with each other via an appropriate routing mechanism.

The system is assumed to be an asynchronous message-passing model [2], [10]. Roughly speaking, an event (such as arrival of a message from an incoming neighbor) triggers a state-transition at each node(such as local computation or outgoing message transmission). Every node follows the specification of the distributed algorithms, i.e., there is no failure. The message may be delayed but every message will *eventually* be delivered to the intended recipient.

B. Problem Definition

XXX

Ruifan's text below

III. Introduction and Setup

Let $\{e_1 \cdots e_l\}$ be a set of distinct items in a distributed system, with unique IDs $\{id_1 \cdots id_l\}$. Let $A_1 \cdots A_p$ be a set of distinct attributes for each record. For every item e_i, the attribute A_j is a non-negative value. Denote the value of attribute A_j of record e_i by $A_j(e_i)$. Denote the sum of all attributes of e_i by $N_i = \sum_j A_j(e_i)$. Denote $N = \sum_i N_i$. Without lost of generality, we assume each attribute rests in a distinct peer. The goal is to find a lightweight algorithm to find the k records whose sum of attributes is the highest. We also want to find the theoretical average performance of our algorithm.

IV. Modification of LightWtTopk

A. The Algorithm

1) Multiply the frequency
   For each $A_j(e_i)$, replace it by $A_j(e_i) - c$ where $c \in (0, \frac{N}{pl})$ is a constant. By doing this, we amplify the relative value of N_i for those $N_i > \frac{N}{l}$, and decrease the relative value of N_i for those $N_i < \frac{N}{l}$.
2) Generating a list of random variables
   In each peer A_j, generate a list of tickets of form $\langle ID, r\rangle$ where ID is a record e_i, and r is a random variable generated from $Exp(A_j(e_i))$,.
3) Pruning the list
   Each peer prunes the list of tickets they have generated. Ticket whose random variable is above a threshold T are discarded. T is known as the sampling threshold.
4) Merging list by minimum
   Each peer exchanges the top L tickets of its pruned list with each of its neighbors. L is known as the propagate count. During the exchange, each peer keeps only the minimum random variable. Notice after the exchange, every peer have tickets with the same value , i.e. the minimum random variable generated by the id.
5) Cropping the merged list
   Each peer sort their merged list in descending order of random variable value, and crops it to have only L topmost records.
6) Running the algorithm multiple times and merging the results.
   Run previous algorithms s times. s is known as the run count of the algorithm. At the end of each run, a list emerges. Obtain a final list according to the generated s lists by counting the number of appearance of a record in the s list. If a record occurs in at lest m out of c lists, the it is included in the final output. m is called the merge threshold.

Parameter	Description
A_j(e_i)	The value of attribute A_j of record e_i
N_i	The sum of all attributes of record e_i
Sampling threshold (T)	Threshold for sampling in the second phase immediately following ticket generation, which also defines the sampling ratio
Propagate count (L)	Number of tickets each agent passes
Cycle count (c)	Number of cycles of ticket exchange;
Run count (s)	Number of runs of the algorithm
Merge threshold (m)	Number of runs in which a record must have appeared as Top-k for it to be included in final Top-k

Table: Notations and parameters of the algorithm

Theorem 1. *By replacing each $A_j(e_i)$ by $A_j(e_i) - c$ where $c \in (0, \frac{N}{mn})$, we amplify the relative value of N_i for those $N_i > \frac{N}{n}$, and decrease the relative value of N_i for those $N_i < \frac{N}{n}$. Furthermore, the higher the original N_i, the higher the amplification effect.*

Proof. We prove the case for $N_i > \frac{N}{n}$. Since $N_i > \frac{N}{n}$, we have

$$N < N_i \cdot n$$
$$cN < cn \cdot N_i$$
$$-cN > -cn \cdot N_i$$
$$N_i \cdot N - cN > N_i \cdot N - cn \cdot N_i$$
$$N \cdot (N_i - c) > N_i \cdot (N - cn)$$
$$\frac{N_i - c}{\sum_{i=1}^{n} N_i - c} = \frac{N_i - c}{N - cn} > \frac{N_i}{N}$$

which means $\frac{N_i}{N}$ has increased after the amplification.
Furthermore, let N_i, N_j be the sum of attributes of e_i, e_j, then since $0 < c < \frac{N}{mn}$, we have

$$N_i > N_j$$
$$\frac{c}{N_i} < \frac{c}{N_j}$$
$$1 - \frac{c}{N_i} > 1 - \frac{c}{N_j}$$
$$\frac{N_i - c}{N_i} > \frac{N_j - c}{N_j}$$
$$\frac{(N_i - c) \cdot N}{(N - cl) \cdot N_i} > \frac{(N_j - c) \cdot N}{(N - cl) \cdot N_j}$$
$$\frac{N_i - c}{N - cl} \cdot \frac{N}{N_i} > \frac{N_j - c}{N - cl} \cdot \frac{N}{N_j}$$
$$\frac{\frac{N_i - c}{N - cl}}{\frac{N_i}{N}} > \frac{\frac{N_j - c}{N - cl}}{\frac{N_j}{N}}$$

which is the desired inequality. □

B. Experiments

We generate nine datasets from Zipfian distribution by varying the following two parameter:

1) Number of items:$[700, 1500, 5000]$
2) The skewness of the distribution: $[1, 1.5, 2]$

In our experiments, we fixed $N = L = c = 5$.We then experimented with nine different settings of (s, m) pairs on each of these nine datasets. Each experiment was repeated 20 times. Finally, the average precision, recall, and list length was computed across the 20 runs for each parameter setting.

C. Results

Here we illustrate the result for one of the datasets. The following graph showed the precision and list length of the dataset of 700 items and 1.5 skewness. We compared the results of our algorithm with the original LightWtTopk algorithm.

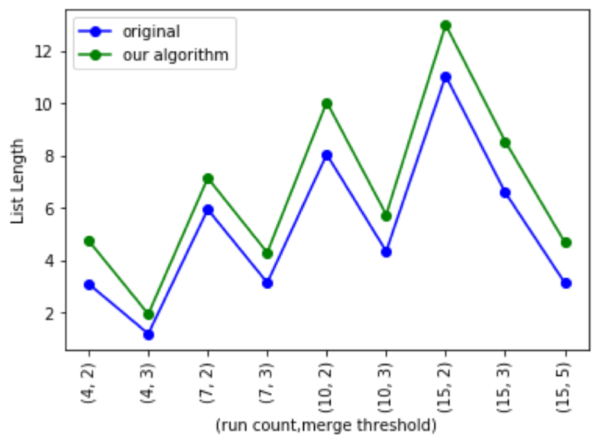

Fig. 1: Comparison of list length

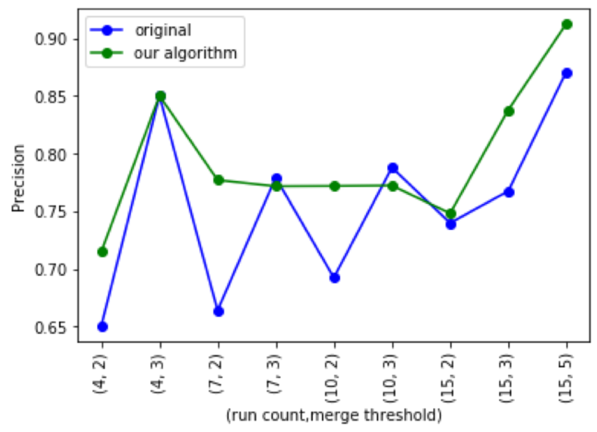

Fig. 2: Comparison of precision

V. Modification of Fagin Algorithm and the Threshold Algorithm

Note that in this section, we introduce faulty nodes into our distributed system. The system may has at most f faulty nodes, which may not report the correct value of the attribute in this peer. Under such system, we introduced the definition of a safe algorithm.

Definition 1. *An algorithm is **safe** if the algorithm will never output an item with no attribute in non-faulty nodes.*

A. Algorithms

Modification of Fagin Algorithm

1) Do sorted access in parallel to each of the m sorted lists L_i. Stop when there are at least k "matches", that is, each of them have been seen in all the lists.
2) For each object R that has been seen: Retrieve all of its fields $x_1, ..., x_m$ by random access. Compute $F(R) = F(x_1, ..., x_m)$. Set $F(x_1, ..., x_m) = 0$ if at least f of the fields are 0.
3) Return the top k answers.

Modification of Threshold Algorithm In this section, we consider the

1) Do sorted access in parallel to each of the m sorted lists L_i. As each item is seen under sorted access: Retrieve all of its fields $x_1, ..., x_m$ by random access. Compute $F(R) = F(x_1, ..., x_m)$. Again, set $F(x_1, ..., x_m) = 0$ if at least f of the fields are 0. If this is one of the top k answers so far, remember it.
2) For each list L_i, let $\hat{xi}$ be the value of the last object seen under sorted access.
3) Define the threshold value t to be $F(\hat{x_1}, \hat{x_2}, \cdots, \hat{x_m})$
4) When k objects have been seen whose grade is at least t, then stop and return the top k answers.

Lemma 1. *The TA always stops at least as early as FA.*

Proof. In Fagin Algorithm, we've seen at least k matches [7], which all have higher grades than the threshold in the TA. □

Again, we see both algorithm will have the same output but the modification of Threshold Algorithm will always be faster asymptotically [7]. Hence in the following context, we only refer to the modification of the Threshold Algorithm.

Theorem 2. *If the aggregate function is monotone, then the algorithm find the top k safe answer.*

Proof. Fagin proved that the Threshold Algorithm output the exact top k items under a fault-free system [7]. Our modification exclude the items that may cause the algorithm to be unsafe. □

Theorem 3. *This algorithm has the best accuracy over all safe algorithm.*

Proof. Suppose there exists a safe algorithm A that achieves better accuracy than our given algorithm in some situation. Then, we know there exist item k with $\geq f$ 0 attributes with a high sum. Consider another situation with the same data distribution with all f nodes of k all being faulty. In this case, algorithm A has an unsafe output, which is a contradiction. □

B. Limitation

Under the realm of safe algorithms, the modification of the Threshold Algorithm perform very well. However in some case, all safe algorithms function badly. Consider the following example with three peers and one faulty node.

	A	B	C
item 1	0	0	1
item 2	0	1	0
item 3	1	0	0

In this case, our previous algorithm does not output any potential candidate for the top items and hence has an accuracy zero. Therefore, we propose some algorithms that allow the occurrence of unsafe items.

VI. Other Unsafe Algorithms

A. Algorithms

Algorithm 1: In the beginning of the algorithm, we randomly picked f peers and assume them to be faulty. Then run

the Threshold Algorithm.

Algorithm 2: Randomly pick one node to exclude from our system for this round. Then, run the Threshold Algorithm to find the top one item among the remaining peer, and add it to the output. Then randomly pick another node. Run the Threshold Algorithm for the remaining peers. Add the highest ranking item that's not in the output yet. Repeat until we have k items.

Algorithm 3: We first find the median of each item over all peers, and then give rank of each item according to their median.

B. Experiments

We run the experiments on both uniformly distributed datasets and Zipfian distributed datasets with 50 peers, 50 items and 5 faulty nodes. For each dataset, we consider the following three faulty behaviors:

1) Zero: The faulty nodes set all the attributes in it to zero.
2) Uniform: The faulty nodes set all the attributes in it to a random number between 0 and $10*$ the maximum value of all attributes.
3) Large: The faulty nodes set all the attributes in it to a random number between $5*$ and $10*$ the maximum value of all attributes.

C. Result

For each three faulty behaviors, we ran all three algorithms with different $k \in [1, 3, 5, 7, 10, 15]$ (that is, different length of output), and for each k, we generated 500 zipfian distributed datasets with parameters 1.34/1.3/1.25 and 50 (items), and for each items, we generated the value of each 50 peers using multinomial distribution. Then, we ran the algorithms in the 500 datasets to calculate the average accuracy. Here we put the result from zipfian distributed datasets with parameters 1.25 and 50 in Figures 3, 4, 5.

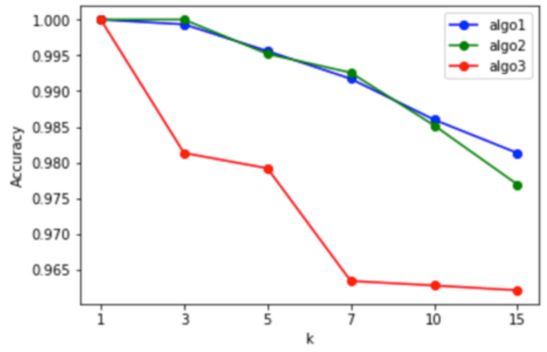

Fig. 3: Zipfian distributed dataset with zero faulty nodes

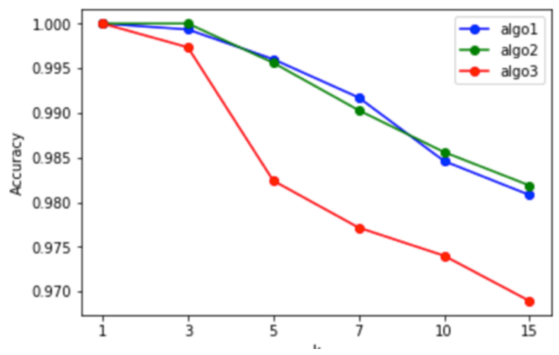

Fig. 4: Zipfian distributed dataset with uniform faulty nodes

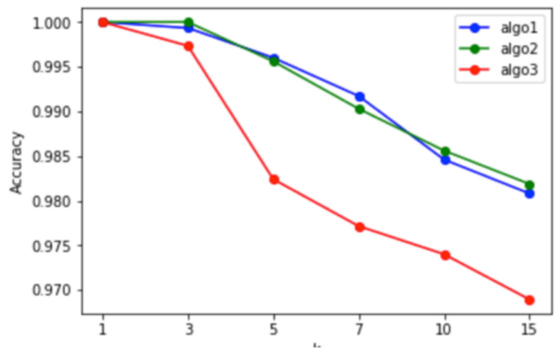

Fig. 5: Zipfian distributed dataset with large faulty nodes

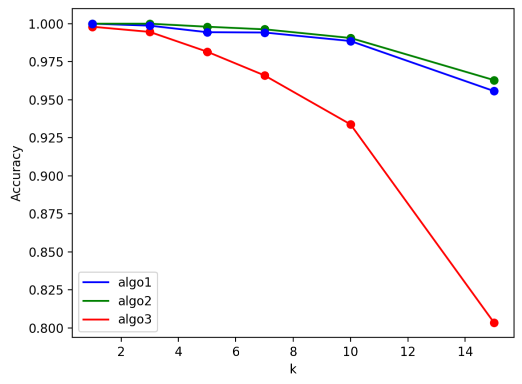

Fig. 6: Zipfian distributed dataset with zero faulty nodes and first parameter 1.34

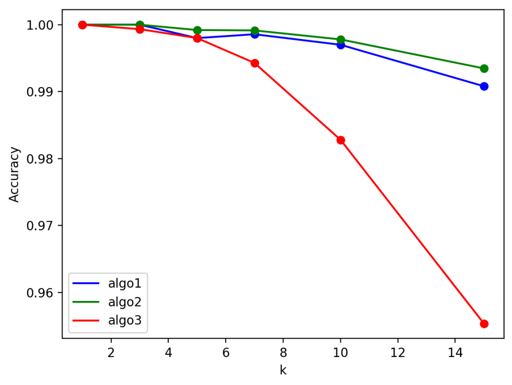

Fig. 7: Zipfian distributed dataset with zero faulty nodes and first parameter 1.25

As we can observe from Figures 3, 4, 5, keeping zipfian parameters and k constant, among the three faulty behaviors, zero faulty behavior will decrease the accuracy the most.

We also observed that keeping k and faulty behavior the same, as first parameter of zipfian approaches 1, the accuracy of each 3 algorithms will increase. We reflect this observation in Figures 6 and 7.

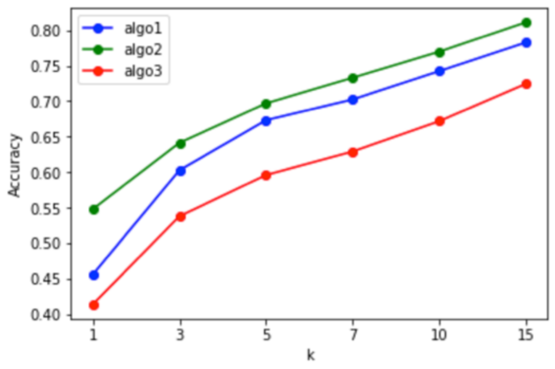

Fig. 8: Uniform distributed dataset with zero faulty nodes

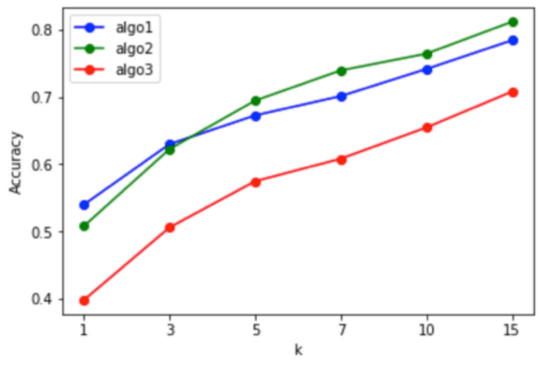

Fig. 9: Uniform distributed dataset with uniform faulty nodes

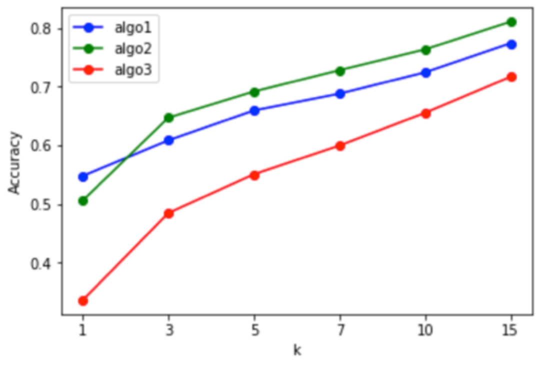

Fig. 10: Uniform distributed dataset with large faulty nodes

We also ran our algorithms using uniform distributed datasets. From Figures 8-10, we can observe that as k increases, each algorithms will have better accuracy in uniform distributed datasets.

D. Real-World Data Testing

Dataset: We use sensor data from Array of Things, a networked urban sensor project. Under subsystem of chemsense, we extract HRF values (short term for human readable values) of pollutant concentration (CO, SO_2, ground-level O_3, and NO_2) in the unit of ppm from Chicago complete daily data sets. We then calculate the AQI using concentration data, the breakpoints table, and linear interpolation equation from the EPA document(https://archive.epa.gov/ttn/ozone/web/pdf/rg701.pdf). According to US EPA, we set the highest AQI value between the 4 pollutants to be the AQI of a particular hour (originally a day).

Faulty behaviors: We take account of three faulty behaviors in the sensor data sheet:

1) Negative: The sensor detects negative HRF value of pollutant concentration which is out of the range between the lower bound and upper bound of human readable value (repeatability of the sensor value is $\pm$ 2%).
2) Large: Under unexpected circumstances, the sensor contacts high concentration of pollutants and temporarily detects a large HRF value which gets AQI $\geq$ 500, the highest breakpoint in the table.
3) Unavailable: A sensor HRF value is marked to ‘NA‘, indicating that it is unavailable. We select the data set without ’NA’ values in our experiment.

Goal: Our unsafe algorithms can predict top 6 hours with worst air quality in early mornings (from midnight to 6AM) of Chicago from 10/12/2019 to 10/18/2019 with a high accuracy. After we exclude all the negative values, we get the correct top 6 hours shown in the graph below. All the AQIs come from SO_2, the contributing pollutant.

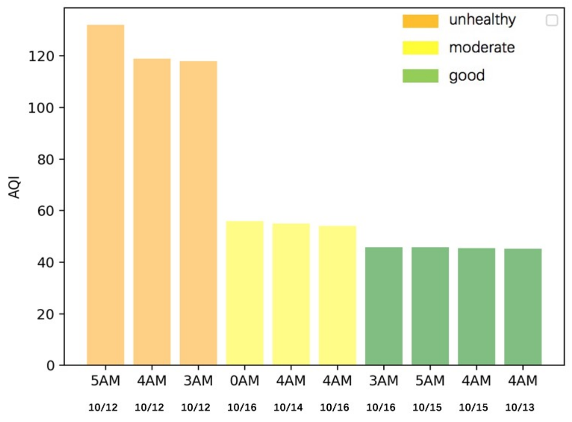

Fig. 11: Top 10 Hours with Worst Air Quality in Early Morning from 10/12 to 10/18 in Chicago

Experiment: On 10 zipfian distributed data sets with 42 items (hours) and 48 peers(AQIs), we randomly pick five sensors to have large faulty behavior. For each value of the faulty sensor, we change the value to a random large AQI value. Then, we run each of the three algorithms on each of the 10 data sets and get the average of the accuracy from 10 data sets for each algorithm with top 1,3,and 6 choices. We then repeat this process for random faulty behavior, that is, either large or negative. Because of the presence of negative nodes, the median is 0 at most cases. So we modify algorithm 3 and calculate the mean instead.

Result:

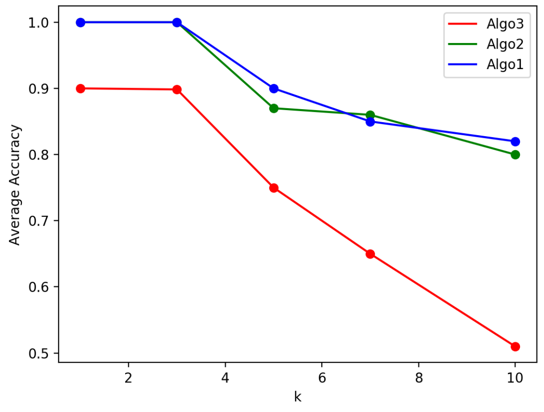

Fig. 12: Large Faulty behavior

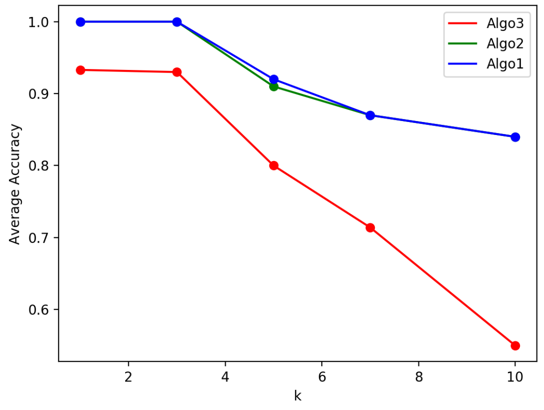

Fig. 13: Random Faulty behaviors

Overall, our algorithms perform well. Large behaviors affect the accuracy more. As a result, by processing real-world sensor data with fault model in simulation, we prove fault-tolerance properties of our unsafe algorithms.

References

[1] D. Amagata, Y. Sasaki, T. Hara, and S. Nishio. Efficient processing of top-k dominating queries in distributed environments. *World Wide Web*, 19(4):545–577, July 2016.

[2] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. Wiley Series on Parallel and Distributed Computing, 2004.

[3] N. Bruno, L. Gravano, and A. Marian. Evaluating top-k queries over web-accessible databases. In *Proceedings 18th International Conference on Data Engineering*, pages 369–380, 2002.

[4] P. Cao and Z. Wang. Efficient top-k query calculation in distributed networks. In *Proceedings of the Twenty-third Annual ACM Symposium on Principles of Distributed Computing*, PODC '04, pages 206–215, New York, NY, USA, 2004. ACM.

[5] W. K. Dedzoe, P. Lamarre, R. Akbarinia, and P. Valduriez. *As-Soon-As-Possible Top-k Query Processing in P2P Systems*, pages 1–27. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[6] V. Deolalikar and K. Eshghi. Lightweight approximate top-k for distributed settings. In *2014 IEEE International Conference on Big Data (Big Data)*, pages 835–844, Oct 2014.

[7] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '01, pages 102–113, New York, NY, USA, 2001. ACM.

[8] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4):11:1–11:58, Oct. 2008.

[9] X. Y. Li, Y. Wang, and Y. Wang. Complexity of data collection, aggregation, and selection for wireless sensor networks. *IEEE Transactions on Computers*, 60(3):386–399, March 2011.

[10] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[11] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 491–502, New York, NY, USA, 2003. ACM.

[12] S. Michel, P. Triantafillou, and G. Weikum. Klee: A framework for distributed top-k query algorithms. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, pages 637–648. VLDB Endowment, 2005.

[13] S. Nath, P. B. Gibbons, S. Seshan, and Z. R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, pages 250–262, New York, NY, USA, 2004. ACM.

[14] B. Patt-Shamir and A. Shafrir. Approximate top-k queries in sensor networks. In *Proceedings of the 13th International Conference on Structural Information and Communication Complexity*, SIROCCO'06, pages 319–333, Berlin, Heidelberg, 2006. Springer-Verlag.

[15] B. Patt-Shamir and A. Shafrir. Approximate distributed top-k queries. *Distributed Computing*, 21(1):1–22, Jun 2008.

[16] M. Theobald, G. Weikum, and R. Schenkel. Top-k query evaluation with probabilistic guarantees. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pages 648–659. VLDB Endowment, 2004.

[17] D. Zeinalipour-Yazti, Z. Vagena, D. Gunopulos, V. Kalogeraki, V. Tsotras, M. Vlachos, N. Koudas, and D. Srivastava. The threshold join algorithm for top-k queries in distributed sensor networks. In *Proceedings of the 2Nd International Workshop on Data Management for Sensor Networks*, DMSN '05, pages 61–66, New York, NY, USA, 2005. ACM.